

[0001] PIPELINE ARCHITECTURE FOR MAXIMUM A
POSTERIORI (MAP) DECODERS

[0002] FIELD OF THE INVENTION

[0003] The present invention relates to the field of processing error detection and correction algorithms. More specifically, the present invention relates to an efficient pipelined architecture for processing maximum a posteriori (MAP) algorithms.

[0004] BACKGROUND OF THE INVENTION

[0005] Some error correction algorithms, such as the Turbo Decoder algorithm, use variations of the MAP algorithm to recover a sequence of information bits from an encoded bit sequence that has been corrupted by noise. The recursive nature of the calculation required by the MAP algorithm makes implementation costly.

[0006] For example, Figure 1 illustrates the sequence output by the MAP algorithm as a function of a set of "forward" metrics, and a set of "backward" metrics. However, each forward metric $\alpha(k)$, is a function of the previous forward metric, $\alpha(k-1)$ and each reverse metric $\beta(k-1)$, is a function of the next reverse metric, $\beta(k)$. As illustrated in the timeline diagram of Figure 1, an architecture that implements this algorithm requires a buffer large enough to hold either all of the forward metrics

or all of the reverse metrics such that the other set of metrics can be calculated while the output is calculated, which design leads to a decoder whose latency is proportional to approximately twice the size of the block that needs to be decoded.

[0007] In an effort to reduce the buffer required by the MAP algorithm, a modified version of the MAP algorithm, called the sliding window algorithm, has been developed. By making a small approximation in the reverse metric calculations, the sliding window approach reduces the size of the required metric buffer. This is accomplished by dividing the received sequence into windows, and then processing each window.

[0008] Figure 2 illustrates a timeline of how the sliding window calculations are performed when the data has been divided into two windows. The length of the tail and learn size are typically very small compared to the amount of data to be processed. It is clear that as long as the window size is relatively large compared to the size of the learning window, the latency through the decoder is not significantly increased but the size of the buffer required to hold the forward metric is significantly decreased.

[0009] Therefore, an objective of the present invention is to reduce both latency and cost associated with implementing such algorithms.

[0010] SUMMARY OF THE INVENTION

[0011] In the pipelined MAP decoder architecture of the present invention, the sliding window approach is modified so that processing time can be decreased. Once the forward metrics have been calculated for the first window, the reverse metrics for each window are calculated while the forward metrics for the next window are calculated. As each new forward metric is calculated and stored into memory, the forward metric from the previous window is read from memory so that the new reverse metric can be calculated. Each forward metric from the previous window is read from memory on the same clock edge that the new forward metric for the next window is written to the same memory location. By reading and writing the forward metrics to memory in this manner, the size of the forward metric buffer does not need to be increased. The pipelined calculations may also be performed if the data is divided into two windows. Although this architecture was developed for a Turbo Decoder, any decoder that uses a version of the MAP algorithm can use it. The pipelined sliding window architecture decreases processing time. The standard sliding window architecture would need to run at a significantly higher clock rate to achieve the same throughput.

[0012] BRIEF DESCRIPTION OF THE DRAWINGS

[0013] The present invention will hereinafter be described in conjunction with the appended drawing figures, wherein like numerals designate like elements and:

[0014] Figure 1 is one example of a time line of a prior art error correction algorithm architecture;

[0015] Figure 2 is a second example of a time line of a prior art error correction algorithm architecture in which the forward and reverse metrics are calculated using sliding windows;

[0016] Figure 3 is a block diagram of a turbo decoder in which the error correction architecture of the present invention may reside;

[0017] Figure 3a is a block diagram of an alternative turbo decoder wherein the calculations of the forward and reverse metrics are reversed; and

[0018] Figure 4 is a time line of the error correction architecture of the present invention.

[0019] DETAILED DESCRIPTION OF THE DRAWINGS

[0020] Figure 3 of the present invention is a block diagram of a turbo Decoder in which the pipeline decoder architecture of the present invention may reside. In the pipelined MAP decoder architecture of the present invention, the sliding window approach is modified so that processing time can be decreased. Figure 4 illustrates the timeline accomplished by the present invention. Once the forward metrics have been calculated for the first window, the reverse metrics for each window are calculated while the forward metrics for the next window are calculated. As each new forward metric is calculated and stored into memory, the forward metric from

the previous window is read from memory so that the new reverse metric can be calculated. Each forward metric is read from memory on the same clock edge that the new forward metric is written to the same memory location. By reading and writing the forward metrics to memory in this manner, the size of the forward metric buffer does not need to be increased.

[0021] Figure 3 shows a block diagram of one embodiment of a turbo decoder incorporating the principles and or techniques of the present invention.

[0022] The turbo decoder 10 receives data from a transmitting facility such as, for example, a base station which converts each data bit ("1" or "0") into three bits, namely a data or systematic bit(s), a first parity bit (p1) and a second parity bit (p2).

The sp1p2 data is applied to a register 12 which also receives extrinsic data read out from an extrinsic memory 14, to be more fully described, hereinbelow, and an address obtained from interleave address register 16. Memory register 12 thus initially receives and stores the sp1p2 data, the extrinsic data appearing at line 14a and an address at which the extrinsic data is located, at line 16a. The address accompanies the sp1p2 data throughout the calculations for a purpose to be more fully described hereinbelow.

[0023] The sp1p2 data is transferred from register 12 to gamma (γ) calculator 18 and is further stored in local memory 20.

[0024] As is conventional in turbo decoders, three quantities, alpha (α), beta (β) and gamma (γ) are defined. For a specific state and a specific time step, α has a

value that defines the probability that the coder is at that state at that specific time step. Alpha is derived recursively starting from time $k=1$ and moving forward in time. The value β is similar to α but works backwards in time. Gamma (γ) is defined as the transition probability that the coder will move from state at a given time to some allowed state at the next subsequent time increment. Alpha (α) can be calculated for all states in a trellis based on the state transition probabilities represented by gamma (γ). The gamma (γ) calculation performed at stage 18 is stored in register 22. Calculation stages 24 and 26 respectively calculate each alpha and normalize the alpha calculations. Each alpha (α) value is calculated based on the input from register 22 as well as the previously calculated alpha value provided at input 24b and outputted from calculation stage 26 through multiplexer 28 and register 30, which holds eight (8) calculated values. The output of register 30 is coupled to the input of alpha memory 32 which stores the first calculated alpha value at the first memory location 23a and also provides the calculated alpha value to input 24b.

[0025] In order to initialize the calculation and, starting at the initial state, the initial eight (8) alpha metrics are set to some initial value, which is applied at the initialized input 28a of multiplexer 28 in order to enable the calculation stages 24 and 26 to calculate the eight (8) values for alpha. As was mentioned hereinabove, the sp1p2 data is stored in local memory 20.

[0026] Initially, all of the alpha values are calculated, whereupon the beta values are calculated by utilization of the sp1p2 data which read in reverse order from

local memory 20 (i.e., "last-in, first-out order") in order to perform the calculations required for the backwards recursive formula for beta. As the sp1p2 data last read in local memory 20 is read into register 34, which contains not only the sp1p2 data, but the extrinsic value (which in the initial stage operation is 0), as well as the data representing the memory location in which the initial extrinsic value located in the extrinsic memory 14. The sp1p2 an extrinsic data undergo calculation at gamma calculation stage 36. The output of the gamma calculation stage 36 is applied to gamma registers 38 and 40. The beta calculations are respectively performed by beta calculation stage 44 and beta normalization stage 46. Initially, a start condition of binary one ("1") is applied to input 42a of multiplexer 42. The normalized beta calculation is initially applied to extrinsic value calculation stage 50 through output register 48 which further applies the last calculated input to input 42b of multiplexer 42. Extrinsic value calculator stage 50 calculates an extrinsic value for each time state k by looking at the alpha value for register 52 received at input 58, the gamma value from register 38 received at input 50b and the beta output from register 48 received at input 50c. Registers 48, 52 and 38 are provided to assure time registration of the signals at the extrinsic value calculator stage 50.

[0027] The intermediate value calculated by first value extrinsic calculator stage 50 is applied to register 54 which transfers its contents to the second stage 56 of the extrinsic value calculator.

[0028] As was mentioned hereinabove, register 34 transfers its contents to register 58 which in turn transfers its contents to register 60, the contents of register 60 being subtracted from the extrinsic value appearing at the output of the second extrinsic value calculation stage 56, this subtraction stage being performed at the subtraction circuit 62.

[0029] The extrinsic value obtained at stage 56 is further applied to a soft-in-hard-out (SIHO) circuitry 64 comprised of a binary state determining circuit 64 receiving the output of second extrinsic calculation stage 56. The operation of circuit 66 in SIHO circuit 64 will be set forth in greater detail hereinbelow.

[0030] The difference output at difference circuit 62 is applied to register 68 which applies the extrinsic value calculation to extrinsic memory 14 at 14b. As was mentioned hereinabove, local memory 20, in addition to storing the data, parity and extrinsic values, further stores the first extrinsic value address of extrinsic memory 14, this address being successively coupled through memory register 34 and time synchronizing registers 70, 72 and 74 to provide the location in extrinsic memory 14 where the extrinsic value calculation is to be stored, the memory location data being applied to extrinsic memory 14 at 14c.

[0031] As was set forth hereinabove with reference to the example shown in Figure 2, half of the calculations for determining alpha are performed during the first time window $k/2$.

[0032] Calculation of the reverse metrics (β) are performed during the last half ($k/2$) of the first window. Alpha values are read out from alpha memory 32 in the reverse order from which they are stored. The alpha values calculated during the forward metric for window 2 (see Figure 4), are simultaneously stored into the memory location from which the alpha values calculated during window 1 are read out for purposes of calculating the extrinsic value, thereby reducing the memory capacity by one half, in the embodiment of Figure 3. It should be noted that the newly calculated alpha values are stored in reverse order from those calculated during the first window.

[0033] In each subsequent pair of window calculations, the number of calculations performed being a function of the number of desired iterations for calculating the extrinsic value, the order of read out and write in of alpha values in alpha memory 32 is reversed so that, as previously calculated alpha values are read out stored in the order of last memory location to first, the alpha values are read out in the reverse order from first memory location to last and the alpha values determined in the window 2 of the second iteration for the forward metric, new values calculated at 24/26 are read in to those locations from which alpha values are being read out.

[0034] As was described hereinabove, when an extrinsic value has been calculated, i.e., upon the completion of the first interration, this extrinsic value is read out of extrinsic memory 14 and is used during the calculations of the next iteration.

Conventional control circuitry, not shown for purposes of simplicity, determines the number of iterations to be performed.

[0035] As was described hereinabove, as each extrinsic value is determined, it is applied to circuitry 66 which determines whether the data bit is a "1" or "0" by examining its amplitude, and when above a certain threshold is determined to be a "1" and when below a certain threshold is determined to be a "0". This established value is applied to register 76 and is merged together with the extrinsic value memory location, derived from register 74 and applied to merge circuit 78. The output bit is written into a memory 84. The SIHO circuitry 64 writes each bit into a memory location in which each row is 16 bits wide. The merge circuit multiplexer 78, multiplexer circuit 80 and output memory read register 82 operate so as to utilize all 16 binary bits of each memory location by storing 16 data bits evaluated by the binary state determining circuit 66.

[0036] Although the embodiment shown in Figure 3 teaches an implementation in which alpha is calculated during the first window and beta is calculated during the latter portion of the first window, it should be understood that the alpha and beta calculations may be reversed, as shown in Figure 5, while still deriving all the benefits of the embodiment shown in Figure 1, namely the significantly reduction in calculation time as well as a 50% reduction in the memory requirements for the turbo decoder of Figure 3 as compared with present day techniques and apparatus. The architecture of the present invention enables further

reductions in memory size. For example, the data may be processed using three (3) windows, four (4) windows, etc, which provide further reductions in memory size.

For example, using four (4) windows results in a memory size that is $\frac{1}{4}$ the memory size compared to processing where no windowing is employed.

[0037] Figure 4 also illustrates how the pipeline calculations would be performed if the data were divided into two windows. Neglecting the size of the learning windows and number of tail bits, the latency through the pipeline sliding window decoder in this example is proportional to $1 \frac{1}{2} K$ as opposed to $2 K$ in the simple sliding window architecture. Latency can be reduced by changing the window size, number of windows, and learning size according to the amount of data that needs to be processed.

[0038] Although the above-described architecture was developed for a turbo decoder, all convolution codes can use an MAP decoder. The calculation of the forward metrics may be calculated before or after the reverse metrics. The reverse metrics could be calculated first, and then the forward metrics can be calculated while the output calculations are performed. This can be accomplished as shown, for example, by the embodiment of Figure 3a wherein calculation block 24^1 is a beta calculator; calculation block 24 is a beta normalization calculation block, memory 32^1 is a beta memory; calculation block 44^1 is an alpha calculation block and calculation block 46^1 is an alpha normalization calculation block.

[0039] The operation of the embodiment of Figure 3a is otherwise substantially the same as the embodiment of Figure 3.

* * *

20200603/001